TITLE:

$\mathbb{Z}/p\mathbb{Z}$上の因数分解と格子算法(数式処理研究の新たな発展)

AUTHOR(S):

山中, 亜希子; 長坂, 耕作

CITATION:

山中, 亜希子 ...[et al]. $\mathbb{Z}/p\mathbb{Z}$上の因数分解と格子算法(数式処理研究の新たな発展). 数理解析研究所講究録 2007, 1572: 59-65

ISSUE DATE:

2007-11

URL:

http://hdl.handle.net/2433/81307

RIGHT:

京都大学学術情報リポジトリ
KURENAI 紅
Kyoto University Research Information Repository

数理解析研究所講究録
第1572巻 2007年 59-65

$\mathbb{Z}/p\mathbb{Z}$ 上の因数分解と格子算法

山中亜希子
AKIKO YAMANAKA
神戸大学総合人間科学科
GRADUATE SCHOOL OF CULTURAL STUDIES AND HUMAN SCIENCES, KOBE UNIVERSITY *

長坂耕作
KOSAKU NAGASAKA
神戸大学人間発達環境学研究科
GRADUATE SCHOOL OF HUMAN DEVELOPMENT AND ENVIRONMENT, KOBE UNIVERSITY †

1 はじめに

$\mathbb{Z}/p\mathbb{Z}$ 上の多項式の因数分解法には，良く知られているものとしてBerlekampアルゴリズムとNiederreterアルゴリズム [2] がある．そのどちらにもGCDを求めるステップがあるが，何ら策を講じなければ，どちらもそのステップには膨大な因子の組合せがあるために時間がかかる．今回の発表は，Niederreiterアルゴリズムにおける，このGCD計算を行い既約因子を求めるステップの改良に関する試験的な取り組みである．具体的には，この部分に格子算法を用いることでアルゴリズムの高速化を目指している．まずは有名な2つのアルゴリズムの概要とGCD計算のステップについて述べてから，今回試みた改良方法についてと，実験結果について述べる．

2 $\mathbb{Z}/p\mathbb{Z}$ 上の因数分解法の概要

因数分解すべき $\mathbb{Z}/p\mathbb{Z}$ 上の多項式を $f(x)$ として，各アルゴリズムの概要を述べる．

2.1 Berlekampアルゴリズムの概要

1. 次式を満たす行列 $Q=(q_{n-i,n-j})\in(\mathbb{Z}/p\mathbb{Z})^{n\times n}$ を構成する．

$$x^{kp}\equiv q_{k,n-1}x^{n-1}+q_{k,n-2}x^{n-2}+\cdots+q_{k,1}x+q_{k,0}\pmod{p,f(x)}$$

2. I を n 次の単位行列として，$Q-I$ の零空間の基底をなす r 個のベクトル $(g_{n-1}^{(i)},\cdots,g_0^{(i)})^t\in(\mathbb{Z}/p\mathbb{Z})^n$ $(i=1,\ldots,r)$ に対応する r 個の多項式 $g(x)^{(i)}=g_{n-1}^{(i)}x^{n-1}+\cdots+g_0^{(i)}$ を構成する．

3. $f(x)$ を $g(x)^{(i)}-s$ $(s\in\mathbb{Z}/p\mathbb{Z})$ とのGCDを計算することにより既約因子に分解する．

*065f739f@stu.kobe-u.ac.jp
†nagasaka@main.h.kobe-u.ac.jp

2.2 Niederreiter アルゴリズムの概要

Niederreiter アルゴリズムも Berlekamp アルゴリズムと同様に，因数分解の問題を線形化することにより既約因子を求めており，次の有理関数体上の微分方程式が用いられている．

$$y^{(p-1)} + y^p = 0$$

この微分方程式は線形方程式となるため，Berlekamp での行列に似た行列を用いることで既約因子を求めていく．

1. 微分方程式に対応する線形方程式の行列 $M_p(f) \in (\mathbb{Z}/p\mathbb{Z})^{n\times n}$ を作成する．
2. I を n 次の単位行列として，$M_p(f) + I$ の零空間の基底をなす r 個のベクトル $(h_{n-1}^{(i)}, \cdots, h_0^{(i)})^t \in (\mathbb{Z}/p\mathbb{Z})^n$ $(i = 1, \ldots, r)$ に対応する r 個の多項式 $h(x)^{(i)} = h_{n-1}^{(i)} x^{n-1} + \cdots + h_0^{(i)}$ を構成する．
3. $f(x)$ を $\sum_{i=1}^{r} s_i h(x)^{(i)}$ $(s_i \in \mathbb{Z}/p\mathbb{Z})$ との GCD を計算することにより既約因子に分解する．

3 GCD の取り方

どちらのアルゴリズムも最後に GCD をとって既約因子を求めるステップがある．Berlekamp の場合，先に書いた総当りで求めるものと，確率的なものがある．また，Niederreiter の場合も，総当りのものや，基底を変換することにより効率よく求めていく方法 [3] によるもの，確率的なもの [1] などがある．

3.1 Berlekamp アルゴリズムの GCD のとり方 (総当り)

Berlekamp アルゴリズムでは図 1 のように，$f(x)$ やその自明でない因子と $g(x)^{(i)} - s$ との GCD を計算することにより既約因子に分解していく．零空間の全ての基底に対応する多項式との GCD を計算することで，既約因子をすべて求めることができる．従って，GCD 計算を最悪の場合に $O(pr)$ 回必要とする．

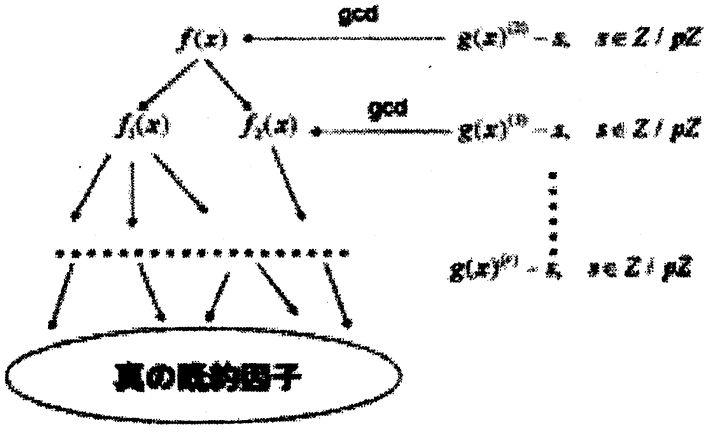

図 1: Berlekamp アルゴリズムでの GCD の計算

3.2 Niederreiter アルゴリズムの GCD のとり方 (総当り)

Niederreiter アルゴリズムでは図 2 のように，$f(x)$ やその自明でない因子と，零空間の基底に対応する r 個の多項式の線形和との GCD を計算することにより既約因子に分解していく．全ての線形和との GCD を計算することで，既約因子をすべて求めることができる．従って，GCD 計算を最悪の場合に $O(p^r)$ 回必要とする．

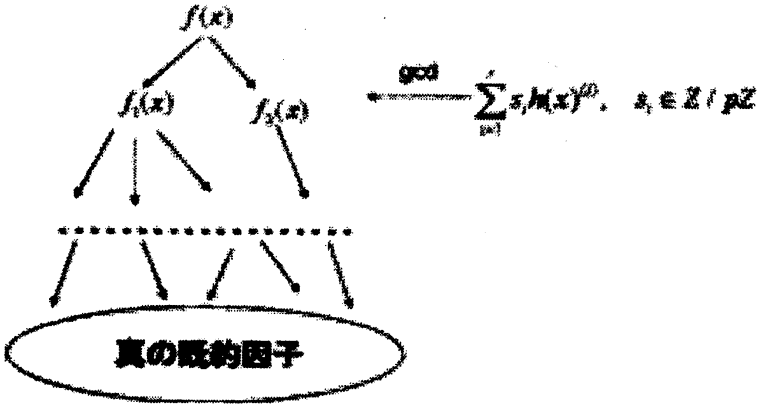

図 2: Niederreiter アルゴリズムでの GCD の計算

この図からもわかるように，自明でない因子に分解することが可能な線形和を構成する s_i の組がわかれば GCD 計算の回数を減らすことができる．後に述べるが，実際に行った試みでは，そのような s_i を比較的簡単に検出することを目指した．

3.3 Niederreiter アルゴリズムの GCD のとり方 (基底変換)

この方法は，H. Niederreiter，R. Gottfert により 1995 年によって発表 [3] されているもので，零空間の基底を元の基底 $B = \left\{\frac{h_1}{f}, \cdots, \frac{h_r}{f}\right\}$ から簡約化した基底 $B_1 = \left\{\frac{u_1}{v_1}, \cdots, \frac{u_r}{v_r}\right\}$ に変換し，アルゴリズム 1「Basic Splitting Step」を用いることで，必要となる GCD 計算の回数を $O(2r-3)$ 回に減らしている．

アルゴリズム 1 (Basic Splitting Step(GCD 計算))
入力： $f(x)$ の因子 $w(x)$

出力： $w(x)$ の既約因子

Step 1. $\gcd(w, v_i)$ を計算し，$w(x)$ の自明でない因子なら $w(x)$ を 2 つの因子に分割し，それぞれの因子に対して *Basic Splitting Step* を再帰的に適用する．

Step 2. $I(w) = \{1 \leq i \leq r : w|v_i\}$ を求め，$\beta \in \mathbf{F}_p$ に対して $\gcd(u_i + \beta\frac{w'v_i}{w}, v_i)$ が自明でない因子なら $w(x)$ を 2 つの因子に分割し，それぞれの因子に対して *Basic Splitting Step* を再帰的に適用する．自明な因子であれば $w(x)$ は既約となる．

4 GCD ステップの改良

総当りの場合，どちらのアルゴリズムもかなりの計算量が必要であるため，それらの改善を行ったアルゴリズムが提案されている．今回の試みもそれらと同じく，Niederreiter アルゴリズムの GCD ステップの改

善の試みとなる．先にも述べた通り，自明でない因子を求めるのに必要な線形和を生成する s_i の組み合わせを格子算法を使って簡単に検出することでの改善の試みである．

具体的な試みの内容は，部分終結式写像と格子算法を用いることで，複数の多項式の線形和との余因子候補を見つけ出す．その際，格子算法による $\mathbb{Z}$ 上の高速因数分解法と同じく，格子算法を $\mathbb{Z}/p\mathbb{Z}$ で適用できるようにすると共に，複数の多項式の線形和に適用するため，Sylvester 行列を拡張する．

部分終結式写像

次の写像 $Syl_r(f,g)$ のことを，$f(x)$ と $g(x)$ の r 次の部分終結式写像という (n を $f(x)$ の次数，m を $g(x)$ の次数とする)．

$$Syl_r(f,g): \begin{array}{ccccc} \mathcal{P}_{m-r-1} & \times & \mathcal{P}_{n-r-1} & \to & \mathcal{P}_{n+m-r-1}, \\ (s(x) & , & t(x)) & \mapsto & s(x)f(x)+t(x)g(x), \end{array}$$

ここで，$r=0,\ldots,\min\{n,m\}-1$ であり，$\mathcal{P}_d$ は d 次以下の多項式全体の集合とする．このとき，この写像のカーネルの要素は $f(x)$ と $g(x)$ の余因子候補となっている．これを複数の多項式の未知の線形結合へ適用することで s_i を求めることを試みた．なお，0 次の部分終結式写像の行列表現が Sylvester 行列であることに注意されたい．

$$u(x)f(x)+t(x)g(x)=0$$

$$u(x)f(x)=-t(x)g(x)$$

$$\Leftarrow \boxed{g(x)=\sum_{i=1}^{r} s_i h(x)^{(i)},\ s_i \in \mathbb{Z}/p\mathbb{Z}}$$

$$u(x)f(x)=-t(x)\sum_{i=1}^{r} s_i h(x)^{(i)}$$

格子算法の利用

自明でない GCD を与える線形和 $\sum_{i=1}^{r} s_i h(x)^{(i)}$ を求めることは，$\mathbb{Z}$ 上の因数分解で発生する試し割りと同じ組合せの問題である．$\mathbb{Z}$ 上の因数分解の場合，格子算法により効率良く組合せを求める方法が提案されており，本発表でも格子算法により $u(x)f(x)=-t(x)\sum_{i=1}^{r} s_i h(x)^{(i)}$ を満たす s_i の組を求めることに取り組んだ．繰り返しになるが，線形和が求まれば $f(x)$ の自明でない因子が見つかることに相当する．

Sylvester 行列の拡張

GCD を計算する多項式を $f(x)=f_n x^n+\cdots+f_1 x+f_0$ と $g(x)=g_m x^m+\cdots+g_1 x+g_0$ とした場合，0 次の部分終結式写像 (即ち，終結式) の行列表現となる Sylvester 行列は，多項式の係数を図 3 のように並べたものになる．なお，Sylvester 行列は畳み込み行列 $C_m(f)$ と $C_n(g)$ で表すことができ，対応する部分を図の左側に記載した．$f(x)$ の k 次の畳み込み行列 $C_k(f)$ とは，$k-1$ 次の多項式との積を行列 $C_k(f)$ と縦ベクトルとの積で表現するのに使われる．

未知の線形結合 $\sum_{i=1}^{r} s_i h(x)^{(i)}$ を求めるために，図 4 の左側の行列のように，零空間の基底に対応する多項式それぞれの畳み込み行列を縦に配置する．その上で，$\mathbb{Z}/p\mathbb{Z}$ 上で格子算法を用いるために右側の行列

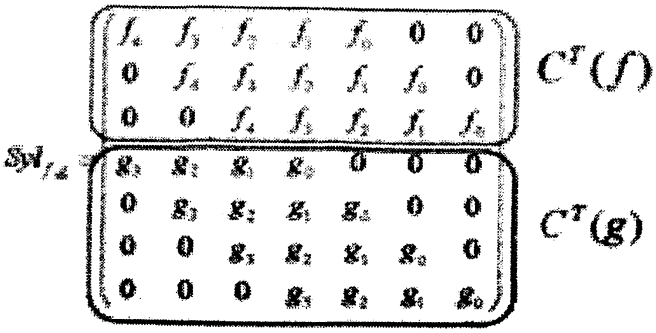

図 3: Sylvester 行列

のように行を追加する．この行列，ないしは更に拡張した行列に格子算法を適用することで，未知の線形結合を求められないかの試みを行っている．

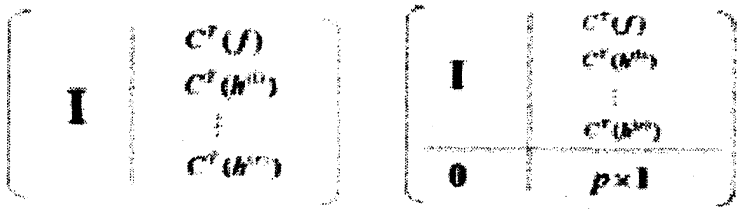

図 4: 行列の構成

4.1 実際の例

これまでに述べたことを実験した例として，$f(x) = x^4 + 2x^3 + 2x \in \mathbb{Z}/3\mathbb{Z}[x]$ の因数分解を考える．Niederreiter アルゴリズムにより行列を求め，その零空間の基底から次の多項式が構成される．

$$\{h_1(x) = x^3 + 2,\ h_2(x) = x^2,\ h_3(x) = x + 2\}$$

Sylverster 行列を拡張したのが左側の行列であり，更に行を追加するなどしたものが右側の行列である．

$$\begin{pmatrix}
0&2&0&2&1&0&0\\
0&0&2&0&2&1&0\\
0&0&0&2&0&2&1\\
2&0&0&1&0&0&0\\
0&2&0&0&1&0&0\\
0&0&2&0&0&1&0\\
0&0&0&2&0&0&1\\
0&0&1&0&0&0&0\\
0&0&0&1&0&0&0\\
0&0&0&0&1&0&0\\
0&0&0&0&0&1&0\\
2&1&0&0&0&0&0\\
0&2&1&0&0&0&0\\
0&0&2&1&0&0&0\\
0&0&0&2&1&0&0
\end{pmatrix}
\Rightarrow
\begin{pmatrix}
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&200&0&200&100&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&200&0&200&100&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&200&0&200&100\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&200&0&0&100&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&200&0&0&100&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&200&0&0&100&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&200&0&0&100\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&100&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&100&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&100&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&100&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&200&100&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&200&100&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&200&100&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&200&100&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&200&100\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&300&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&300&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&300&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&300&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&300&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&300
\end{pmatrix}$$

このようにして得られた右側の行列に格子算法 (LLL アルゴリズム) を適用した結果が，次の行列である．

$$\begin{pmatrix}
0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 100 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 100 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 100 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 100 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 100 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 100 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 100 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

この行列から s_i の組がわかる．この行列の前から 3 列が $f(x)$ に対する $h_1(x)$, $h_2(x)$, $h_3(x)$ の未知の線形結合の余因子，次の 4 列が $h_1(x)$ に対する $f(x)$ の余因子，その次の 4 列が $h_2(x)$ に対する $f(x)$ の余因子，その次の 4 列が $h_3(x)$ に対する $f(x)$ の余因子となる可能性のある多項式の係数となっている．この例の場合，6 行目から s_i の組を求めることができ，$s_1 = 1$, $s_2 = -1$, $s_3 = 0$ となる．これから線形和を求めると，次の多項式が求まる．

$$\sum_{i=1}^{r} s_i h(x)^{(i)} = 1 \cdot h_1(x) - 1 \cdot h_2(x) + 0 \cdot h_3(x) = x^3 - x^2 + 2$$

この多項式と $f(x)$ との GCD を再度計算するか，上記の 6 行目から直接求めることで，$f(x)$ の自明でない因子が求まる．

$$\gcd(f(x), \sum_{i=1}^{r} s_i h(x)^{(i)}) = x^3 + 2x^2 + 2$$

5 まとめ

今回の試みでは，行列を使ったシンプルな方法で線形和を求めることに取り組んでいる．実験においては，格子算法による結果から未知の線形結合を構成する s_i の組を簡単に見つけられたが，確実に求まるはまだわかっていない．だが，この方法では線形独立な零空間の基底を扱うため，行列の作成がうまくできれば，一定の条件下においては証明ができるのではないかと思われる．また，証明できれば，GCD 計算において今までと違ったアルゴリズムを提案できるかもしれないし，確実に求められることが証明されれば旧来の方法より早いかを，Niederreiter らの基底変換の方法との比較で調べてみたい．

6 発表後に判明したこと

本発表で取り組んだ試みは，発表後の追試や研究により，自明でない因子を求めるのに必要な未知の線形結合を確実に検出することはできないことが判明している．本発表は都度言及していたように，格子算法の活用による Niederreiter アルゴリズムの GCD ステップの改善の試みであり，あくまでも 1 つの試みが失敗しただけであり，引き続き可能性を探っていきたい．

参 考 文 献

[1] S. Gao and J. von zur Gathen. Berlekamp's and Niederreiter's polynomial factorization algorithms. In *Finite fields: theory, applications, and algorithms (Las Vegas, NV, 1993)*, volume 168 of *Contemp. Math.*, pages 101–116. Amer. Math. Soc., Providence, RI, 1994.

[2] H. Niederreiter. A new efficient factorization algorithm for polynomials over small finite fields. *Appl. Algebra Engrg. Comm. Comput.*, 4(2):81–87, 1993.

[3] H. Niederreiter and R. Göttfert. On a new factorization algorithm for polynomials over finite fields. *Math. Comp.*, 64(209):347–353, 1995.